AMATH 482/582: HOMEWORK 2

KATIE HIPPE

AMATH Department, University of Washington, Seattle, WA
kahippe@uw.edu

Abstract. A humanoid robot moves in three distinct patterns, samples of which are recorded and able to be analyzed to find xyz-coordinates of its joints. We have used PCA modes to project these coordinates to a lower dimension and created an algorithm that will recognize the currect movement in real-time, then tested this algorithm for accuracy.

1. Introduction and Overview

Scientists have been tirelessly working on the newest version of a humanoid robot, OptimuS-VD, an impressive feat of engineering that knows how to perform three distinct movements: walking, jumping, and running. While it moves, we wish to be able to immediately classify it into one of these three. We have five samples of each movement, each containing 100 samples for 100 time steps. However, this data has high dimensionality, as it contains xyz-coordinates of each of the robot's 38 joints, for a total of 114 separate dimensions.

We wish to shrink this data into lower dimensions while keeping each movement separate. Then we may categorize each movement much more quickly, ideally without losing accuracy. Using these categories, we may classify data on the robot's movements as one of the three in real-time.

We may do this using PCA: first we examined what each movement looks like in $k-$PCA space for various different values of k. Then we determined how many modes we needed to keep a high level of accuracy on our classification. Finally, we tested our classification system on new data to evaluate how well it worked.

2. Theoretical Background

Our initial movement data was given to us in several files, each with a matrix of 114×100. Within this matrix, the first dimension records the x, y, and z location of each of the robot's 38 joints, yielding 114 unique spatial dimensions. The second dimension is 100 time steps breaking down each 1.4 second recording of each movement. We were given 5 samples of each of the 3 movements, yielding 15 total samples with which to develop our algorithm.

To reduce the dimensionality of our data, we conducted a Principal Component Analysis (PCA) to find the most important components of our data. To do so, we used Singular Value Decomposition (SVD) (1):

$$X = U\Sigma V^T \tag{1}$$

where X is our training data matrix, in this case consisting of an aggregate matrix of all our samples discussed previously. It has been decomposed into three component matrices: U, the left singular vectors, V, the right singular vectors, and Σ, a diagonal matrix filled with the singular values of X sorted in descending order.

Date: February 28, 2025.

Once we have found U using the entirely of our training data, we may use it to project a given data sample z (all xyz coordinates, taken at one time-step) onto the determined PC modes:

$$z = U^T z$$

where we have as many PC modes as spatial dimensions in our initial data. However, in the case of our data this means 114 PC modes, and we wished to reduce the dimensions of our data. So instead we may project on $k-$PC modes, so instead of 114 modes we only have k, using (2)

$$z_k = U_k^T z \tag{2}$$

where U_k^T has the first k column vectors kept intact while the column vectors from $k+1$ and onward are set to 0. When we apply this to a data sample z, the result will be a reduced-dimension in $k-$PCA space, which we may compare with other data in the same $k-$PCA space.

To classify our data, we computed the centroid in $k-$PCA space for all the samples of each motion (3):

$$\text{centroid}(X^k) = (\frac{1}{N}\sum_{i=1}^{N} X_{i,1}^k, \frac{1}{N}\sum_{i=1}^{N} X_{i,2}^k, ..., \frac{1}{N}\sum_{i=1}^{N} X_{i,k}^k) \tag{3}$$

which will result in a coordinate with k dimensions corresponding to the location of the centroid of our data matrix projected onto $k-$PC modes, X^k. Here, k is our number of PC modes and N is the number of samples.

3. Algorithm Implementation and Development

To develop and run these functions and algorithms, we used Python 3.12.8 in Visual Studios Code. For general numerical methods, we used NumPy [1]. For additional numerical methods, we used SciPy [4]. For PCA, scaling, and clustering, we used scikit-learn [3]. For plotting, we used Matplotlib [2].

First, we fit and transformed our training data into PCA-space, looping through $k = 1$ to $k = 114$ to see how many PC-modes we needed to accurately represent our original data. We visualized the output for $k = 2$ and $k = 3$.

Then, we looped again from $k = 1$ to $k = 114$: each time, we calculated the centroid of each type of motion. Then, we evaluated each time step of our training data and placed it within a category of motion based on how close it was to the three centroids. We did the same for each time step of the testing data. (For an extra step, we clustered and labeled testing and training data using **kmeans**). Finally, we calculated the accuracy of the classification based on our previously known truth.

At the end of our $k-$loop, we evaluated the overall accuracy for both testing and training data as a function of k.

4. Computational Results

First and foremost, we scaled our data, as this is good practice as it will reduce any potential biases and ensures a comparable scale. Our first task involved applying PCA such that the PCA modes were spatial modes and the coefficients were time-dependent. We did this for a range of k values, as k denotes our number of PCA spatial modes. For each k, we calculated the Frobenius norm of projected matrix in $k-$modes, and divided by the Frobenius norm of our original data to see how much of our original data was preserved. The results are shown in Figure 1 and listed in Table 1.

Based on this analysis, we see that we only need 7 PCA modes to approximate our original X_{train} with 95% accuracy. We also note that we only need 2 or 3 PCA modes to reconstruct our data with a very high degree of accuracy as well. To visualize what our data looks like in this lower dimension space, we plotted PC1 and PC2 coordinates in Figure 2a and PC1, PC2, PC3

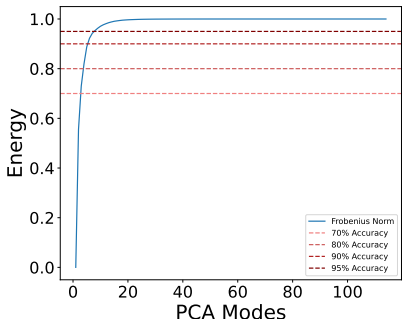


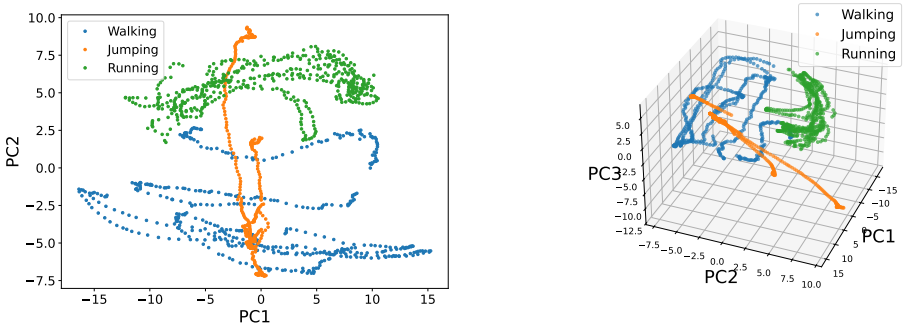
FIGURE 1. PCA Energy vs Modes

% Accuracy	k (PCA modes)
70%	2
80%	3
90%	5
95%	7

TABLE 1. A table

coordinates in Figure 2b. In each case the 'walking' data is plotted in green, 'jumping' is plotted in orange, and 'running' is plotted in blue.

Note that Figure 2b has been rotated to better showcase the different clusters of our data. In both graphs, we see three patterns of movement. The 'jumping' data has a much different shape in both graphs, as it represents a simple up and down movement, while 'walking' and 'running' both have more spatial variation. However, the 'running' has the most spatial variation, which follows from running being a faster activity. Since we have already color-coordinated these three motions based on our ground truth, it is easy to separate them by eye. But if we were given either of these graphs without the aid of colored-coding, figuring out our three groups would be difficult due to the overlap, especially of 'jumping' data overlapping with both 'walking' and 'running.'



(A) 2D Trajectories (B) 3D Trajectories

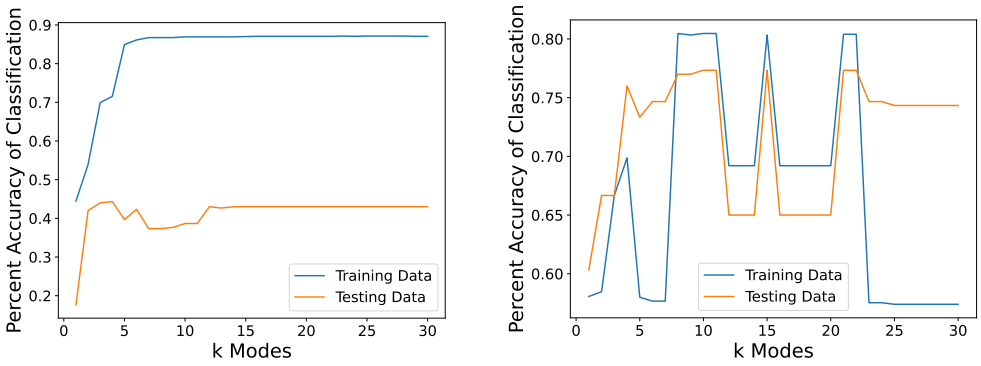
FIGURE 2. Projected X_{train} in 2D and 3D Trajectories

So, to classify these three types of motion, we may have to turn to higher-dimension PCA trajectories. First, we label each sample in our X_{train} based on which motion it originally came from. This yielded a ground-truth vector with which to compare the outputs of our classification algorithm. We may then separate our three data sets within $k-$PCA space and, from the aggregate data, decide upon a particular identifier. Our first option is the centroid, as calculated in (3). This yields a unique coordinate for each type of data.

Our second option utilized a clustering method. Instead of separating our data into 'walking,' 'jumping,' and 'running' and then calculating a coordinate, we let the data divide itself up into three categories using $k-$means clusering. In this algorithm, we chose the number of clusters, always 3, and then randomly chose 3 cluster centers. These centers were then updated until the

sum of distances between data points and centers is minimized. This algorithm yields a label for each data point in our X_{train}, which we then may compare with our ground-truth.

With these processes defined, we ran through a series of k values: For each k value, we determined the mapping U_k^T (see 2) based on all our training data. We then transformed all our training and testing data according to that same U_k^T into $k-$PCA space. We then determined both the centroids and $k-$means cluster centers from our training data. We then individually evaluated each time step from both the training and testing data, classified them according on which center they were closest to, and finally evaluated how accurate our methods had been. These results are graphed for the centroid method in Figure 3a and the clustering in Figure 3b.



(A) Classified with Centroids (B) Classified with K-Means

FIGURE 3. Accuracy of Classification vs k-PCA modes

Note the differing scale of these graphs, as well as their truncation at $k = 30$. Past this point, the graphs do not change significantly.

First, we'll discuss the centroid classification. According to this analysis, the optimal k value to accurately classify the training data is $k = 23$, while the testing data was most accurately classified when $k = 4$. The training data required a relatively high k to achieve this optimal accuracy, most likely due to how many different data points we had and the level of overlap between them. When considering the nature of the three motions, there is a possibility of the robot being in exactly the same position, despite it being in the middle of different actions. We already saw that we need less than 10 PC modes to accurately reconstruct our data to 95%, but we need more PC modes to predict the different types of motion because of the overlap between the robot's positions. Even as we continue to increase k, the accuracy caps out just before 90%.

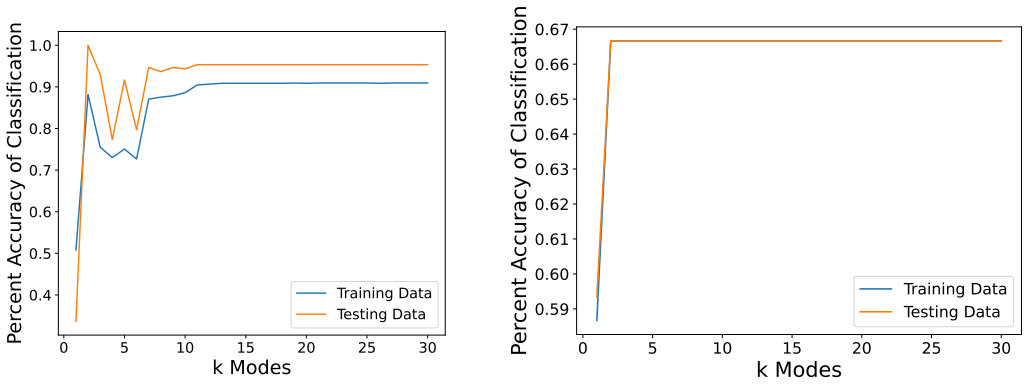
The testing data, when classified according to the same methods as the training data, had a significantly lower accuracy score, barely above random classification. This is in part due to the fact that we've scaled our data, as when we rerun the model without scaling either our training or testing data, we have a much higher accuracy score. However, scaling data is good practice and a way of standardizing our results, so for our main evaluation we will continue looking at the scaled results.

We may now turn our attention to the accuracy when classified with $k-$means. In this case, we have a much different shape of our line, most likely due to the complication nature of our classifications. When this method determined the cluster centers, it does so without knowing which sample came from which data set originally, so there is a high chance of misidentifying samples, especially in lower dimensions. As we saw from Figure 2, there is much overlap between data points. Overall, both the testing and the training data had lower accuracy scores, but they were much more comparable with each other, unlike the centroid method.

Additionally, after k increases beyond our $k = 30$ threshold, the accuracy remains low, only

properly identifying about 50% of our data. Another issue comes in the method we used to calculate our cluster centers: we used a constant initial random state, which might not be the optimal for each value of k, and could negatively impact our results.

As we attempt this same analysis with centered, but not scaled data, we significantly change the shape of the graph:



(A) Classified with Centroids (B) Classified with K-Means

FIGURE 4. Accuracy of Classification vs k-PCA modes on Unscaled Data

Note again the differing scales. Thus scaling the data does end up affecting our analysis, as scaling reduces the variance of our data set and thus changes how accurate our classification ends up becoming. Our optimal k values also change, as now the optimal number of PC modes for the testing data is 2, while the optimal number for the training data has decreased from 23 to 18. However, this could be simple luck: the testing data has a lot fewer instances than the training data, so it is a lot easier to get a higher percentage accurate without necessarily meaning anything about how good the algorithm actually is. However, we will follow Figure 3 as reference for evaluating these algorithms as it is good practice to scale data to reduce the impact of outliers.

5. Summary and Conclusions

Over the course of this project, we developed an algorithm to classify the current time step of the robot's movement into one of its three known actions. To do so, we reduced the dimensions of location data of the robot's joints so as to come to our conclusions quickly and accurately. Using a comprehensive PCA analysis, we determined the optimal number of PCA modes to evaluate the movement. In the future, we may try more alternate methods of classification to hopefully increase our accuracy score, or conduct a different type of PCA analysis based on POD, for example.

Acknowledgements

The author is thankful to our TA Rohin Gilman to help debugging and discussions of the theoretical meaning of testing/training data. We also thank our peers Kaillah Selvaretnam, Heidi Neuman, Kate Everling, Quinn Kelly and Laura Pong for conceptual help surrounding the problem and graphing our results.

References

[1] C. R. Harris, K. J. Millman, S. J. der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, SebastianBerg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.